Costs in software development

Expected costs in a traditional software development project:

- Development costs
- Project management
- UI/UX design
- Quality assurance

Technical debt in software development

Technical debt or design debt:

- Cost of rework caused by poor design

[1] https://vincentdnl.com/drawings/

Hidden technical debt in ML systems

Machine Learning: "The high-interest credit card of technical debt"[1].

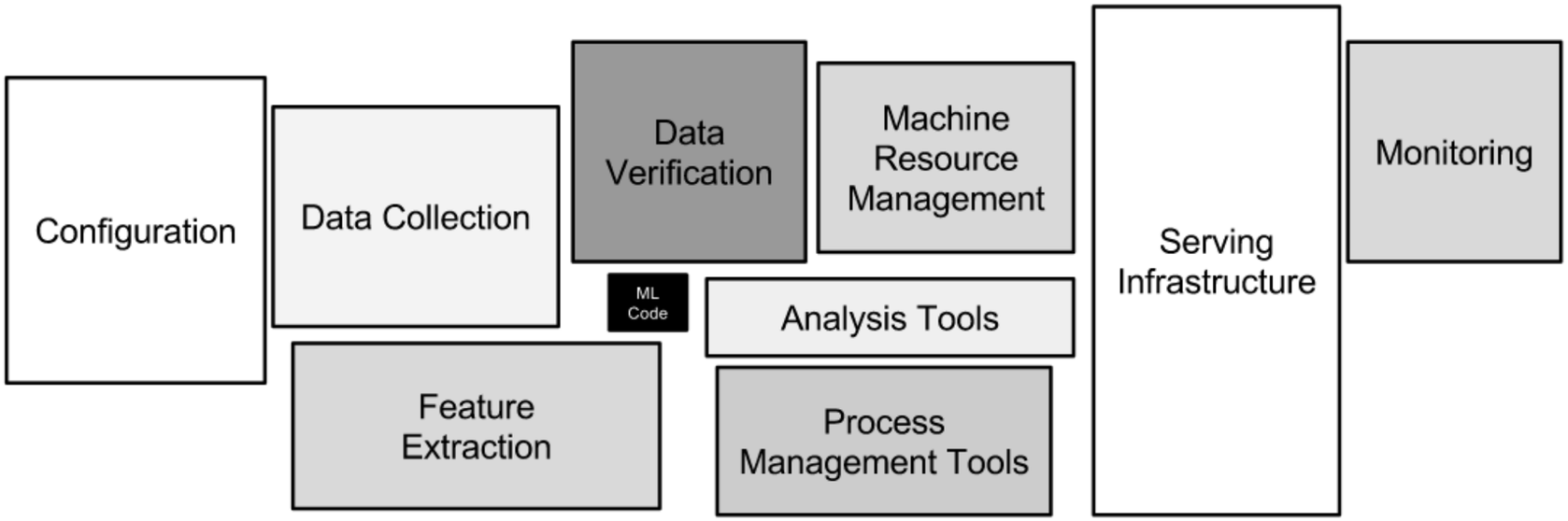
Hidden technical debt can be related to:

1. The data used to train the ML models
2. The models powering the ML system
3. The infrastructure used by the ML system
4. The monitoring of the ML system

[1] https://research.google/pubs/pub43146/

The high-interest credit card of technical debt

ML systems can be complex and become unruly.



MLOps: The best-known way to pay

If ML is the high-interest credit card of technical debt, MLOps is the best way to pay for it.

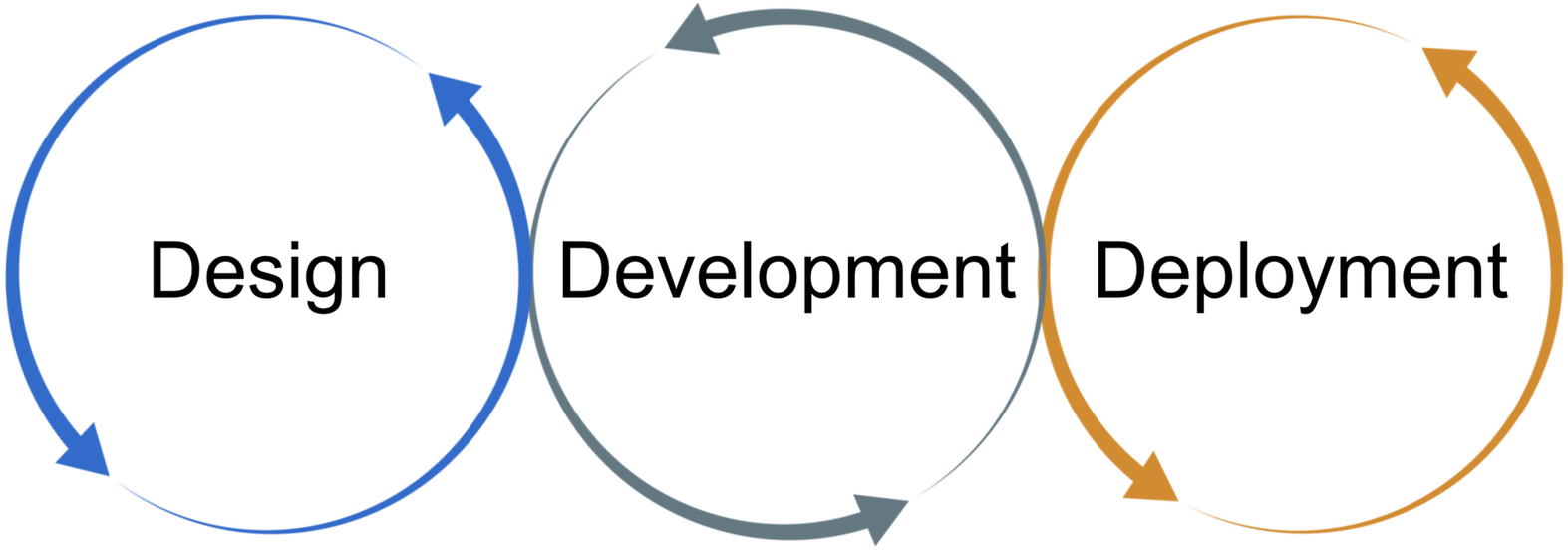
MLOps can include:

- Automated testing
- Automated experiment tracking
- Automated monitoring

To keep the technical debt to a minimum

The MLOps lifecycle

The MLOps lifecycle includes three core stages:

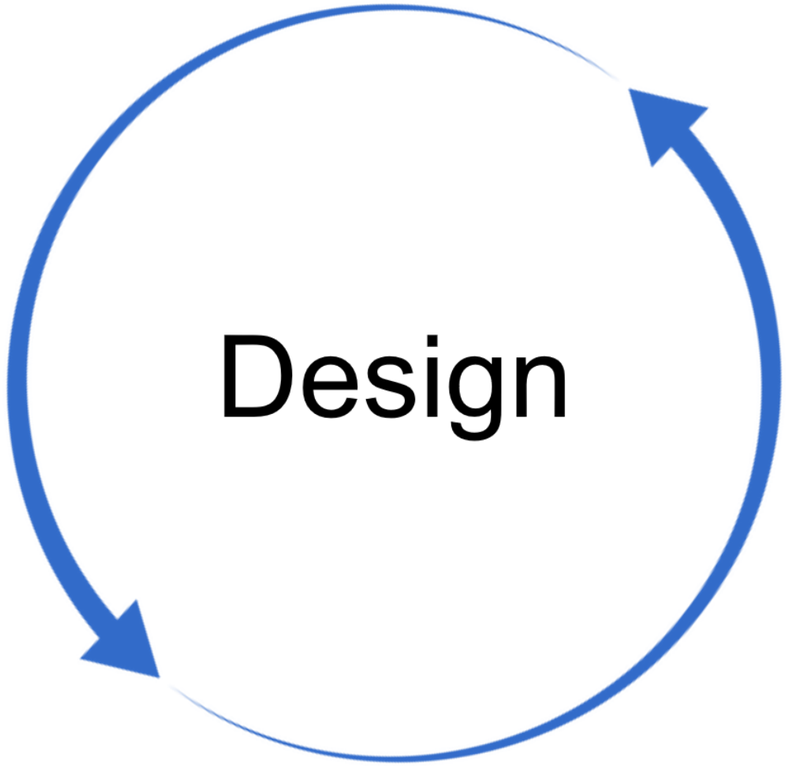


- The three stages are iterative
- The three stages are interconnected and rely on each other
- It is normal to go back and forth between stages

MLOps in the ML lifecycle - Design

The Design stage includes:

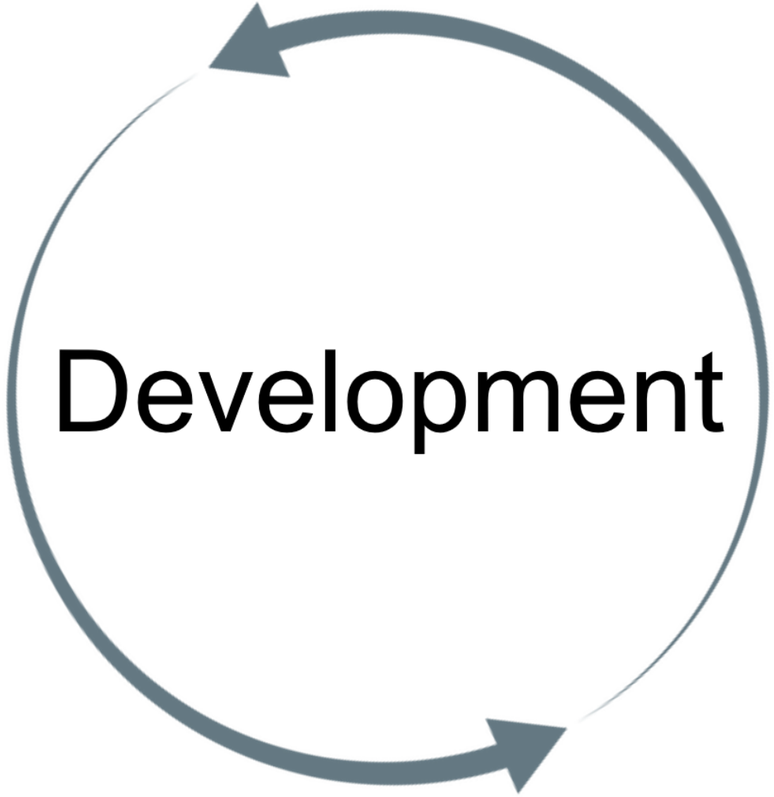
- **BUSINESS UNDERSTANDING**
- **DATA UNDERSTANDING**
- **DESIGNING THE ML SOLUTION**



MLOps in the ML lifecycle - Development

The ML experimentation and development stage includes:

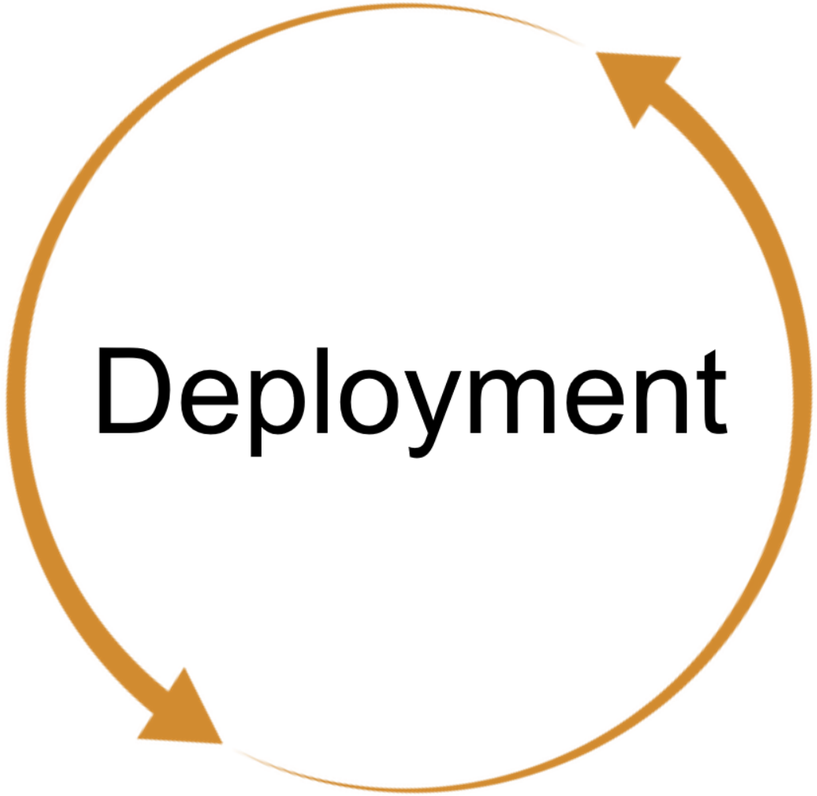
- **DEVELOPING PROOF-OF-CONCEPTS (POCS)**
- **DATA ENGINEERING**
- **MODEL DEVELOPMENT**



MLOps in the ML lifecycle - Deployment

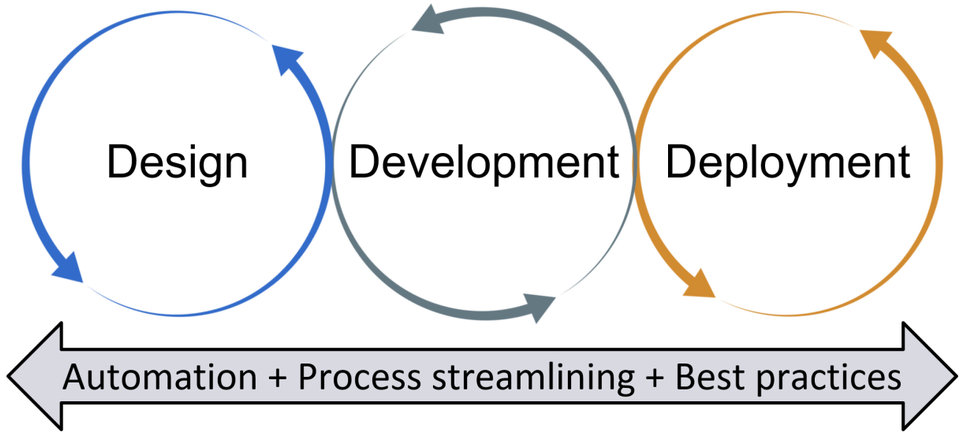
The ML deployment and operations phase include:

- Productionization
  - Testing
  - Versioning
  - Continuous delivery
  - Monitoring



Building for scale: Automation first

We use process streamlining, best practices, and automation



When no automation, process streamlining:

- CRISP-DM
- TDSP [1]

[1] https://www.ibm.com/docs/en/spss-modeler/saas?topic=dm-crisp-help-overview

Process streamlining and best practices

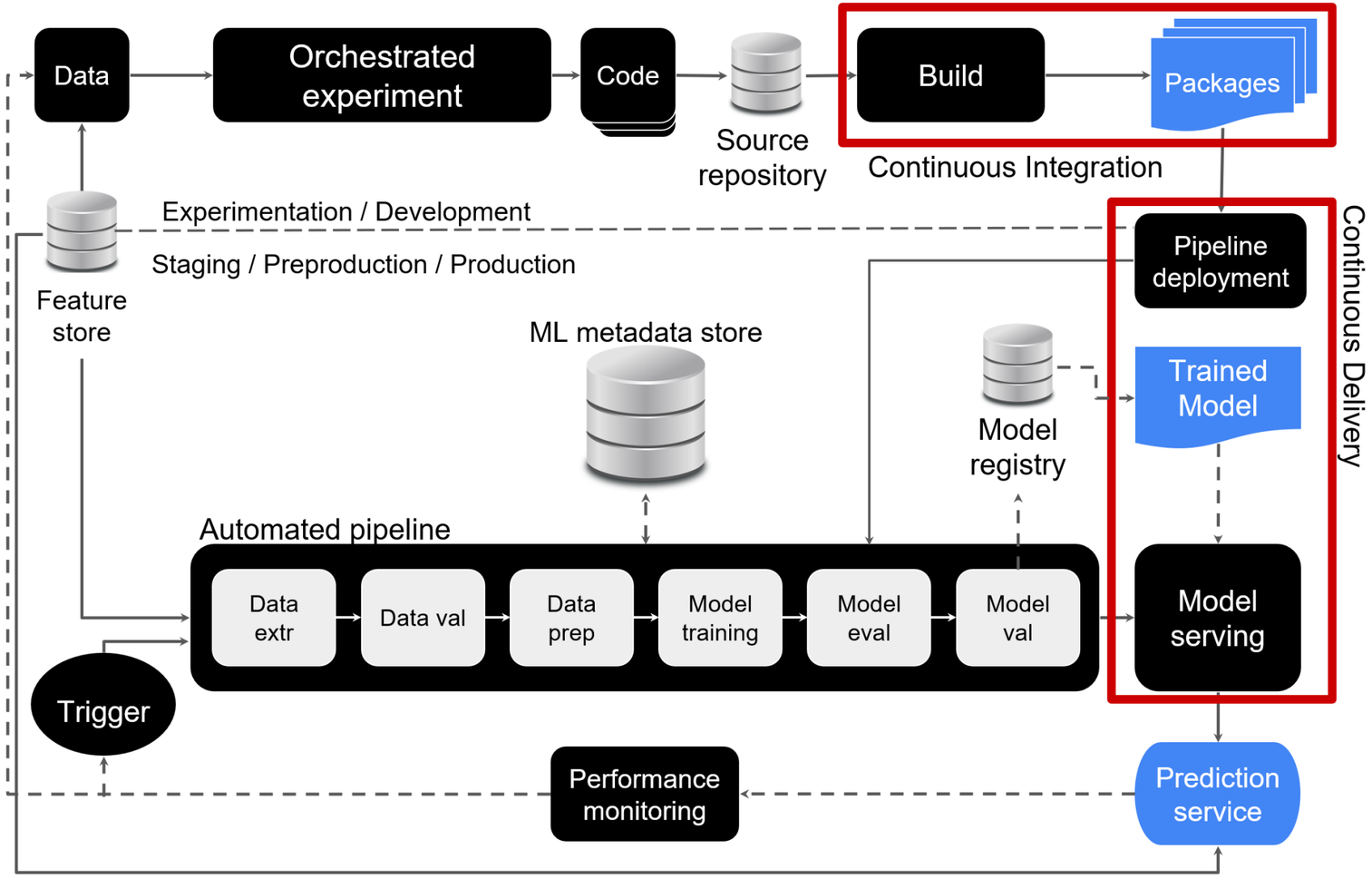
DESIGN PHASE

- Best practices
  - Include domain expertise
  - Involve business stakeholders
  - Get feedback from end-users

DEVELOPMENT PHASE

- Best practices
  - Write clean code
  - Document our work

Fully automated MLOps architecture



[1] https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning

What is MLflow?

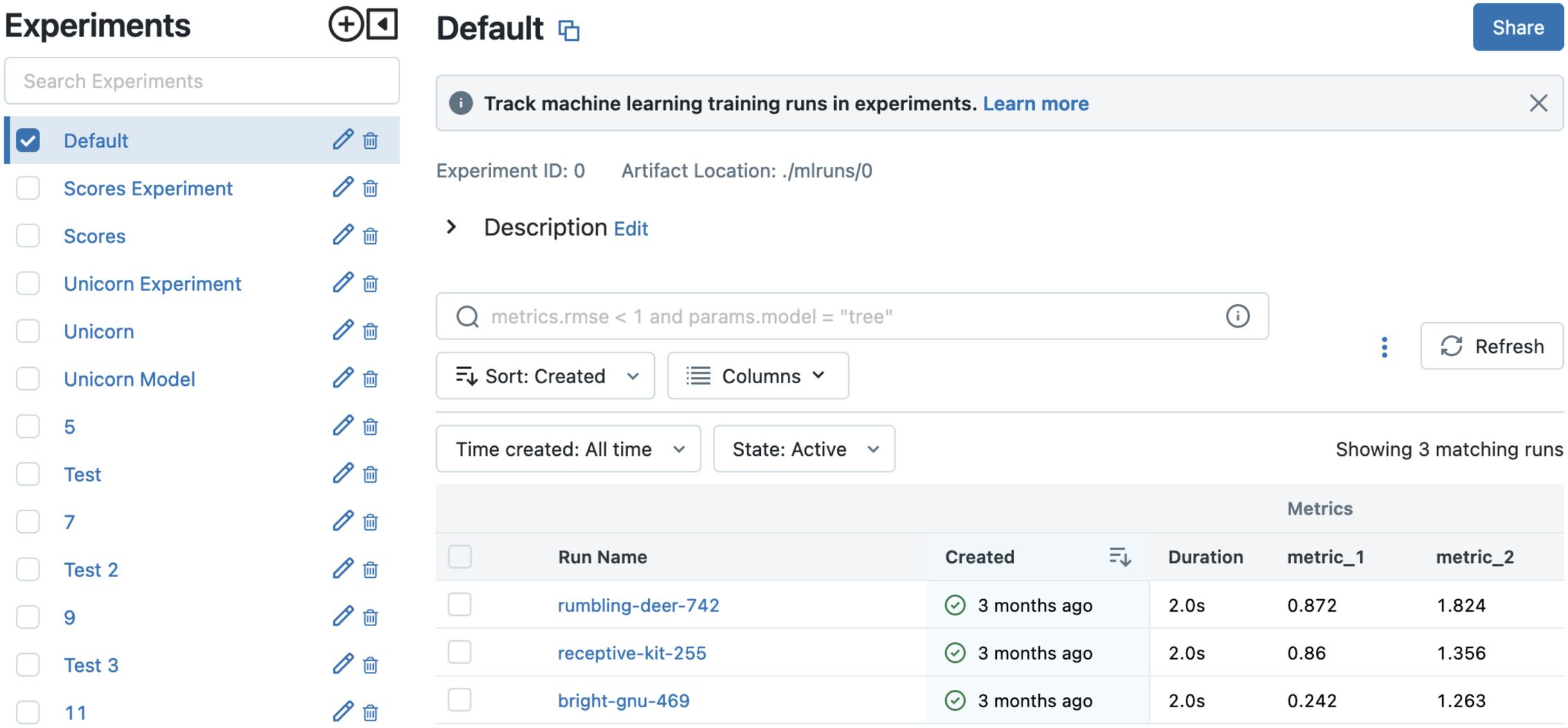
"An open source platform for the machine learning lifecycle" - **MLflow.org**

[1] www.mlflow.org

Components of MLflow

- **MLflow Tracking:**
  - Record metrics and parameters from training runs
  - Query data from experiments
  - Store models, artifacts and code
- **MLflow Models:**
  - Standardize models for deployment
  - Build customized models
- **Model Registry:**
  - Store and version ML models
  - Load and deploy ML models
- **MLflow Projects:**
  - Package ML code for reproducibility
  - Package ML code for repeatability

MLflow experiments



Starting a new experiment

```
import mlflow
# Create new Experiment
mlflow.create_experiment("My Experiment")
# Tag new experiment
mlflow.set_experiment_tag("scikit-learn", "lr")
# Set the experiment
mlflow.set_experiment("My Experiment")
```

Starting a training run

```
import mlflow


# Start a run
mlflow.start_run()
```

```
<ActiveRun: >
```

```
# End a run
mlflow.end_run()
```

Logging to MLflow Tracking

- **Metrics**
  - `log_metric("accuracy", 0.90)`
  - `log_metrics({"accuracy": 0.90, "loss": 0.50})`
- **Parameters**
  - `log_param("n_jobs", 1)`
  - `log_params({"n_jobs": 1, "fit_intercept": False})`
- **Artifacts**
  - `log_artifact("file.py")`
  - `log_artifacts("./directory/")`

Autolog

```
# Automatically log model and metrics
mlflow.FLAVOR.autolog()
```

```
# Scikit-learn built-in flavor
mlflow.sklearn.autolog()
```

Built-In Flavors

- Python Function (`python_function`)
- R Function (`crate`)
- H_2O (`h2o`)
- Keras (`keras`)
- MLeap (`mleap`)
- PyTorch (`pytorch`)
- Scikit-learn (`sklearn`)
- Spark MLlib (`spark`)
- TensorFlow (`tensorflow`)
- ONNX (`onnx`)
- MXNet Gluon (`gluon`)
- XGBoost (`xgboost`)

[1] mlflow.org

- Write custom tools from ML libraries
- Flavors simplify the new for custom code

```
# Import flavor from mlflow module
import mlflow.FLAVOR
```

Open MLflow UI

```
# Open MLflow Tracking UI
mlflow ui
```

Go to: http://localhost:5000

MLflow REST API

[1] istock.com

Model API functions

```
# Save a model to the local filesystem
mlflow.sklearn.save_model(model, path)
```

```
# Log a model as an artifact to MLflow Tracking.
mlflow.sklearn.log_model(model, artifact_path)
```

```
# Load a model from local filesystem or from MLflow Tracking.
mlflow.sklearn.load_model(model_uri)
```

Load model

- Local Filesystem - `relative/path/to/local/model` or `/Users/me/path/to/local/model`
- MLflow Tracking - `runs:/<mlflow_run_id>/run-relative/path/to/model`
- S3 Support - `s3://my_bucket/path/to/model`

Custom Python models

- Built in Flavor - `python_function`
- `mlflow.pyfunc`
  - `save_model()`
  - `log_model()`
  - `load_model()`

Custom model class

- Custom model class
  - `MyClass(mlflow.pyfunc.PythonModel)`
- PythonModel class
  - `load_context()` - loads artifacts when `mlflow.pyfunc.load_model()` is called
  - `predict()` - takes model input and performs user defined evaluation

Example custom Class

```
import mlflow.pyfunc


# Define the model class
class CustomPredict(mlflow.pyfunc.PythonModel):
    # Load artifacts
    def load_context(self, context):
        self.model = mlflow.sklearn.load_model(context.artifacts["custom_model"])
    # Evaluate input using custom_function()
    def predict(self, context, model_input):
        prediction = self.model.predict(model_input)
        return custom_function(prediction)
```

Saving and logging a custom model

```
# Save model to local filesystem
mlflow.pyfunc.save_model(path="custom_model", python_model=CustomPredict())
```

```
# Log model to MLflow Tracking
mlflow.pyfunc.log_model(artifact_path="custom_model", python_model=CustomPredict())
```

Loading custom models

```
# Load model from local filesystem
mlflow.pyfunc.load_model("local")
```

```
# Load model from MLflow Tracking
mlflow.pyfunc.load_model("runs:/run_id/tracking_path")
```

Model Evaluation

- `mlflow.evaluate()` - Performance based on a dataset
- Regression and Classification models

Evaluation Example

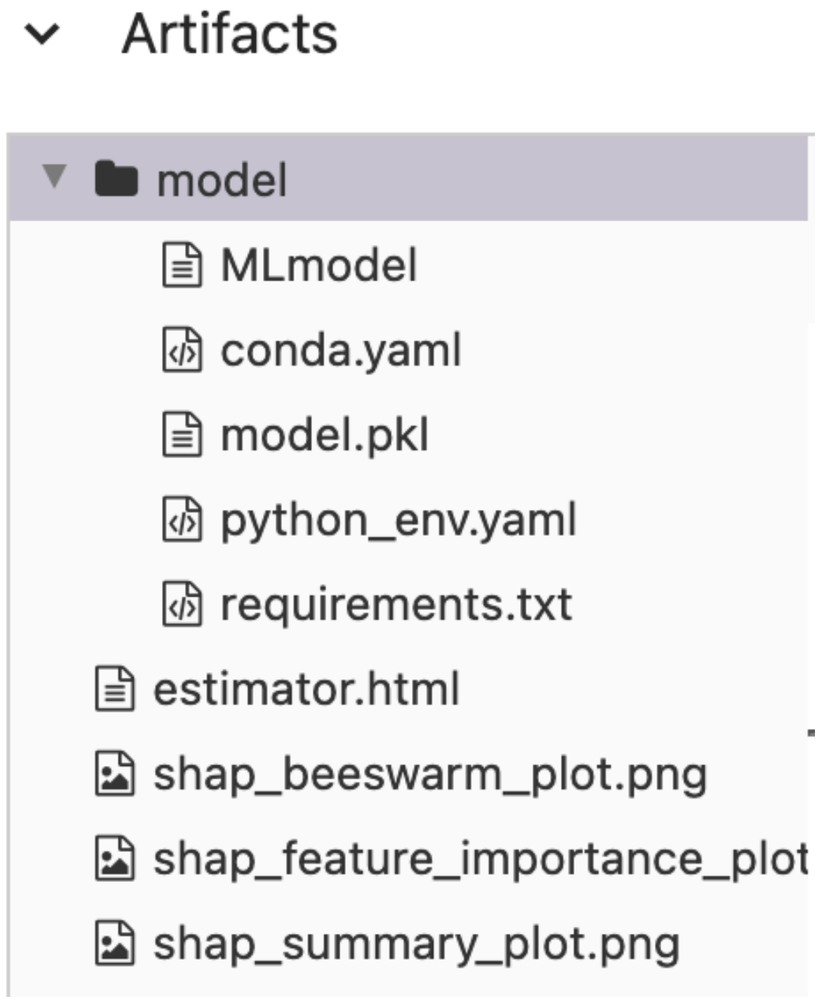
```
# Training Data
X_train, X_test, y_train, y_test = \
    train_test_split(X, y,
    train_size=0.7,random_state=0)

# Linear Regression model
lr = LinearRegression()
lr.fit(X_train, y_train)
```

```
# Dataset
eval_data = X_test
eval_data["test_label"] = y_test

# Evaluate model with Dataset
mlflow.evaluate(
    "runs:/run_id/model",
    eval_data,
    targets="test_label",
    model_type="regressor"
)
```

Tracking UI



SHAP

[1] shap.readthedocs.io

Serving Models

```
# MLflow serve command
mlflow models serve --help
Usage: mlflow models serve [OPTIONS]
```

Serve uri

```
# Local Filesystem
mlflow models serve -m relative/path/to/local/model
```

```
# Run ID
mlflow models serve -m runs:/<mlflow_run_id>/artifacts/model
```

```
# AWS S3
mlflow models serve -m s3://my_bucket/path/to/model
```

Serve example

```
# Serve model from run
mlflow models serve -m runs:/e84a122920de4bdeaedb54146deeb429/artifacts/model
```

```
2023/03/12 16:28:28 INFO mlflow.models.flavor_backend_registry:
Selected backend for flavor 'python_function'
2023/03/12 16:28:28 INFO mlflow.pyfunc.backend: === Running command
'exec gunicorn --timeout=60 -b 127.0.0.1:5000 -w 1 ${GUNICORN_CMD_ARGS} --
mlflow.pyfunc.scoring_server.wsgi:app'
[2023-03-12 16:28:29 -0400] [48431] [INFO] Starting gunicorn 20.1.0
[2023-03-12 16:28:29 -0400] [48431] [INFO] Listening at: http://127.0.0.1:5000
(48431)
[2023-03-12 16:28:29 -0400] [48431] [INFO] Using worker: sync
[2023-03-12 16:28:29 -0400] [48432] [INFO] Booting worker with pid: 48432
```

REST API

- `/ping` - for health checks
- `/health` - for health checks
- `/version` - for getting the version of MLflow
- `/invocations` - for model scoring
- Port 5000

Invocations endpoint

`/invocations`

```
No,Name,Subject
1,Bill Johnson,English
2,Gary Valentine,Mathematics
```

`Content-Type` : `application/json` or `application/csv`

```
{
    "1": {
        "No": "1",
        "Name": "Bill Johnson",
        "Subject": "English"
    },
    "2": {
        "No": "2",
        "Name": "Gary Valentine",
        "Subject": "Mathematics"
    }
}
```

CSV format

- Pandas Dataframe
- `pandas_df.to_csv()`

JSON format

- `dataframe_split` - pandas DataFrame in split orientation
- `dataframe_records` - pandas DataFrame in records orientation

DataFrame split

```
# Dataframe split orientation
{
  "dataframe_split": {
      "columns": ["sex", "age", "weight"],
      "data": [["male", 23, 160], ["female", 33, 120]]
  }
}
```

Invocations Request

```
# Send dataframe_split orientation payload to MLflow
curl http://127.0.0.1:5000/invocations -H 'Content-Type: application/json' -d '{
  "dataframe_split": {
      "columns": ["sex", "age", "weight"],
      "data": [["male", 23, 160], ["female", 33, 120]]
  }
}'
```

```
{"predictions": [1, 0]}
```